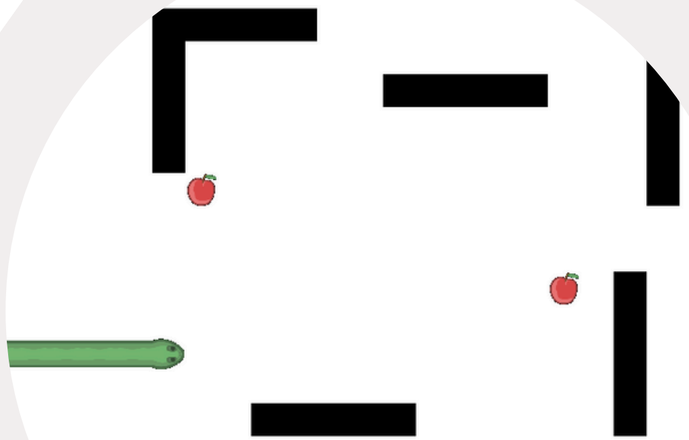
TPG - SNAKE

Inteligência Artificial

João Viegas nº113144
Jorge Domingues nº113278
João Monteiro nº114547



Introduction

- The objective of this project is to develop an intelligent agent capable of autonomously playing the classic Snake game. The agent is designed to analyze the game environment, make decisions, and execute optimal strategies to achieve the best possible results.

Agent's Architecture

- **SnakeDomain**
  - Defines how the snake moves, what states are valid, and how to compute the next state from a given action.
  - Implements a cost function and a heuristic to guide searches.
  - Checks whether a state satisfies the goal.

- **Search Nodes and Tree**
  - Each node holds a state, a reference to its parent, and tracking information like cost and heuristic.
  - The tree expands possible moves, storing unvisited states in open_nodes.
  - Selects and expands the most promising node until finding a path to the goal.

- **Agent**
  - Receives the current game state from the server (map, snake body, food position, etc.).
  - Constructs a search problem using *SnakeDomain* and invokes *SearchTree* to get an action plan.
  - Sends the next move to the server and repeats the process whenever the goal changes or the plan is exhausted.

Algorithm and Decision-Making

- **Movement and Pathfinding**
  - The snake uses **A*** search to reach the food optimally. Each time the snake detects a new piece of food, it constructs a SnakeDomain (providing the current direction, body positions, map, etc.), then runs a SearchTree with the **A*** strategy.
  - The snake prioritizes movement toward spaces marked as 0 on the map, which represent unexplored areas.
  - Using the *sumZeros* function, the agent calculates the number of zeros surrounding each potential movement direction. This encourages exploration of unknown regions while ensuring it avoids confined spaces.
  - When no unexplored areas (zeros) are available in the vicinity, the agent cleans the map using *clearMap*, resetting previously explored spaces to 0. This allows the snake to consider those areas as unexplored again, effectively restarting the search process in the same environment.

- **Behavior Around Food**
  - When food is detected, the agent attempts to find an optimal path using **A*** search.
  - Before moving toward the food, it evaluates the situation:
    - It uses a **greedy search** to assess if it can safely return to a safe point after collecting the food.
    - If it determines that collecting the food would leave it trapped, the agent ignores the food and prioritizes safer actions, such as heading toward its tail or exploring other regions.

- **Safe Points**
  - Safe points are generated dynamically by dividing the map into quadrants and selecting coordinates with enough space to avoid obstacles.
  - These points are marked as 7 on the map and serve as fallback locations if the snake is at risk of becoming trapped or cannot find viable paths.
  - When no other safe options exist, the snake uses these points to navigate safely.

- **Updating and Cleaning the Map**
  - The snake continuously updates its internal map based on the game's current state. It detects and records areas like empty spaces, walls, obstacles, and other players or enemies.
  - If the snake becomes trapped in an area without unexplored spaces, it cleans the map, marking all previously explored areas as unexplored. This allows the agent to restart its exploration process.
  - Additionally, after eating a total of three foods, the map is cleaned to refresh the snake's perspective and ensure it doesn't rely on outdated information.

- **Fallback Strategy**
  - When no food or safe paths are available, the snake defaults to exploring areas with the highest number of unexplored spaces or moves toward its tail as a last resort.
  - To ensure the snake's survival, we use greedy search to verify if it is possible to traverse from the next position to the closest safe point. This evaluation allows the agent to preemptively check the safety of its moves. If it determines that the next position cannot lead to a safe point, it selects an alternative valid direction where the following position avoids immediate risks.

- **Foods Decision**
  - In a single-player game, when the snake's range exceeds 5 and traverse is True, it begins ignoring superfoods because these foods are associated with luck, and the snake opts to prioritize safer strategies. Toward the end of the game, the snake takes the risk of eating all the superfoods in quick succession, now that their positions are known.
  - In multiplayer, the snake stops eating superfoods when the range is greater than or equal to 4 and traverse is True, as this strategy ensures the snake's survival by avoiding unnecessary risks specific to this type of game.

- **Conclusion for the Algorithm**
  - The algorithm implemented is highly effective, allowing the snake to maximize food collection by utilizing a combination of A* for optimal pathfinding and greedy search for safety evaluation. This combination ensures the snake can balance exploration and survival efficiently in most scenarios. However, its performance is less effective when the range is as small as 2, as the snake's movement becomes less systematic and struggles to cover the map uniformly. In contrast, with larger ranges (5 or more), the algorithm excels, as the snake has a broader field of view, enabling better planning and decision-making.

Benchmarks

Test nº	Old Version	New Version
1	31	88
2	33	112
3	34	121
4	47	125
5	48	130
6	49	133
7	64	143
8	77	159

While comparing our version implemented until the November 30th with the most recent version, several significant improvements were identified:

1. **Decision-Making for Direction Changes:**
   - In the earlier version, the snake only changed direction after fully exploring an entire row or column. While systematic, this approach was inefficient as it failed to adapt dynamically to the game's environment.
   - The current version prioritizes unexplored areas by always moving toward the direction with the highest number of unexplored zones. This change significantly enhances exploration efficiency and adaptability.
2. **Incorporation of Safe Points and Risk Analysis:**
   - The previous implementation lacked mechanisms such as safe points or comprehensive risk analysis. This limited the snake's ability to evaluate whether pursuing a piece of food was safe or if it would lead to entrapment.
   - The latest version introduces safe points and employs greedy search to determine if a potential move can safely lead to a fallback location. This enhancement allows the snake to make more informed and strategic decisions, particularly when navigating complex environments.

Observations to our latest version:

In cases where the results are lower, this can be attributed to the snake spending a significant amount of time in the game with a smaller range and failing to consume a superfood that could have increased its range. Another contributing factor could be that the traverse mode was set to False during the game, which hinders the efficiency of the search and limits the snake’s ability to explore the map effectively.

Future Work

- For future improvements, we could explore a more efficient algorithm tailored for a range of 2. One potential approach would involve the snake systematically searching column by column across the map. This strategy could make the snake more efficient, as it is optimized for scenarios with a reduced field of view. By adopting a more uniform search pattern, the snake could better cover unexplored areas, increasing its effectiveness in navigating the environment and finding food even with limited visibility.

Conclusion

- In conclusion, the project was successfully implemented, resulting in an intelligent snake agent capable of autonomously navigating the game environment with strategies suited to different scenarios, including single-player and multiplayer modes. The agent effectively balances exploration, food collection, and survival by leveraging algorithms like **A*** for optimal pathfinding and **greedy search** for safety checks.